AD-753 670

GEOMETRIC MODELING

C. Stephen Carr

Utah University

Prepared for:

Rome Air Development Center
Advanced Research Projects Agency

August 1972

RADC-TR-69-248
Final Technical Report
August 1972

GEOMETRIC MODELING

University of Utah

Sponsored by
Defesne Advanced Research Projects Agency
ARPA Order No. 829

Approved for public release;
distribution unlimited.

Rome Air Development Center
Air Force Systems Command
Griffiss Air Force Base, New York

77

UNCLASSIFIED
Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author) Computer Science University of Utah Salt Lake City, Utah 84112	2a. REPORT SECURITY CLASSIFICATION Unclassified
	2b. GROUP

3. REPORT TITLE

GEOMETRIC MODELING

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)
Technical Report

5. AUTHOR(S) (First name, middle initial, last name)
C. Stephen Carr

6. REPORT DATE August 1972	7a. TOTAL NO. OF PAGES 69	7b. NO. OF REFS 12
8a. CONTRACT OR GRANT NO. AF30(602)-4277	9a. ORIGINATOR'S REPORT NUMBER(S) TR 4-13	
b. PROJECT NO. ARPA Order No. 829		
c. Program Code Number: 6D30	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report) RADC-TR-69-248	
d.		

10. DISTRIBUTION STATEMENT
This document has been approved for public relase and sale; its distribution is unlimited.

11. SUPPLEMENTARY NOTES Monitored by Rome Air Development Center (EMIIO) Griffiss Air Force Base, New York 13440	12. SPONSORING MILITARY ACTIVITY Advanced Research Projects Agency Wash DC 20301

13. ABSTRACT

A system is presented for modeling three-dimensional objects such as buildings and mechanical devices, in the computer. The system has been implemented as an interactive design system. Shaded, half-tone photographs of objects designed with the system are included:

An algorithm is presented for quickly determining if two objects, created by the human user, interpenetrate one another. The user is informed of such violations by the system. Additionally, basic concepts of computer graphics modeling are presented for the benefit of readers not conversant with the terminology of computer graphics.

The implementation of these ideas as an interactive design system for architects has been a formidable task on the 1108. A description of this task is included as an Appendix. The description should be of interest to systems programmers, however, as the implementation used the most modern programming techniques, such as a compiler/compiler to generate a graphics programming language processor, and a hashed associative data base.

I a'

DD FORM 1 NOV 65 1473

UNCLASSIFIED
Security Classification

UNCLASSIFIED
Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Computer Graphics						
Computer Aided Design						
Data Structures						
Computer Modeling						

[illegible]

UNCLASSIFIED
Security Classification

GEOMETRIC MODELING

C. Stephen Carr

University of Utah

This research was supported by the Advanced Research Projects Agency of the Department of Defense and was monitored by David A. Luther, RADC (ISCE), GAFB, NY 13440 under contract AF30(602)-4277.

I.N

FOREWARD

This interim report describes research accomplished by Computer Science of the University of Utah, Salt Lake City, Utah, for the Advanced Research Projects Agency, administered by Rome Air Development Center, Griffiss Air Force Base, New York under Contract AF30(602)-4277. Secondary report number is TR 4-13. Mr. David A. Luther (EMIIO) is the RADC Project Engineer.

This technical report has been reviewed and is approved.

[illegible]

Approved: DAVID A. LUTHER
Project Engineer

ACKNOWLEDGEMENTS

The ideas of Geometric Modeling, as developed in this thesis, are principally those of the author. However, the ideas spring from work on an architectural design system. No less than fifteen people were involved in some way on this project.

Thanks are in order for Professor Stephen L. Macdonald[8] and Dr. Robert Wehrli[11] of the Architecture and Physchology Departments and to the graduate architectures and psychologists who worked under them: Max J. Smith, Ted F. Smith, Jay A. Shadel, and John Archea.

David A. Luther, a fellow computer science graduate student and friend, interfaced his Electrical Design System[5] to the basic geometric modeling program. As the first person to attempt this, he encountered many difficulties between our two programs, which were straightened out in the process. The similar task of connecting the Structural Analysis program went much more smoothly as a result of his labor, which is appreciated.

I wish especially to thank my programmers, who labored with a computer system totally inadequate for interactive computing: J. Robert Geertsen, Sherian Erdman, and David T. Blackburn. The Univac 1108 and its batch processing monitor are so unsuited to the task that it never should have been undertaken.

Finally, I wish to thank my advisor, Dr. David C. Evans, and my wife, Lynne, who played the obvious and necessary roles. Their contribution need not be explained or emphasized to anyone who has struggled through a graduate program.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

ABSTRACT

A system is presented for modeling three-dimensional objects, such as buildings and mechanical devices, in the computer. The system has been implemented as an interactive design system. Shaded, half-tone photographs of objects designed with the system are included.

An algorithm is presented for quickly determining if two objects, created by the human user, interpenetrate one another. The user is informed of such violations by the system. Additionally, basic concepts of computer graphics modeling are presented for the benefit of readers not conversant with the terminology of computer graphics.

The implementation of these ideas as an interactive design system for architects has been a formidable task on the 1108. A description of this task is included as an Appendix. The description should be of interest to systems programmers, however, as the implementation used the most modern programming techniques, such as a compiler/compiler to generate a graphics programming language processor, and a hashed associative data base.

OVERVIEW

Solving a problem on a computer amounts to (1) constructing a model in a representation the computer can work with; and (2) constructing procedures to interactively modify the model. Geometric Modeling is the creation of abstract representations of three-dimensional world objects in the computer. Such modeling was motivated by the construction of a computer-aided design system for architects. Buildings, furniture and mechanical devices such as machinery are examples of objects which require a three-dimensional model in the computer as a basis for their manipulation.

Many objects are easier to model in computer terms than are three-dimensional objects. For example, electronic circuit diagrams are easier, since they are highly stylized, descrete abstractions of real world objects. A transistor is satisfactorily represented by a number of attributes, one of which is its circuit schematic symbol (figure 1). This abstract schematic is what circuit designers need. The architect and mechanical designer want rich detail, instead of abstraction. Detailed, three-dimensional objects are difficult to model and difficult to display on the CRT.

Modeling complicated three-dimensional real-world objects is difficult, and relatively little has been done in this area. Previous work has advanced general data structures for modeling heirarchically defined objects.[2,4,9] The present work concerns the decisions necessary to adapt the general structure to three-dimensional objects.

A number of geometric solids are provided as basic object elements in lieu of points, lines and planes. The other forms are easily derived from this basic form, when necessary. The designer works at a line drawing scope and sees the edges of surfaces and volumes. This is a limitation of present-day technology. Research is underway to provide a raster (half-tone) picture to interact with. In the meantime, the user can produce a shaded picture of this design on an auxiliary scope in about thirty seconds[10].

Examples are given to show how the resulting model works in practice. The modeling methods described here have been implemented as a part of an interactive graphics system. A simple design done with the system is shown in figure 2.

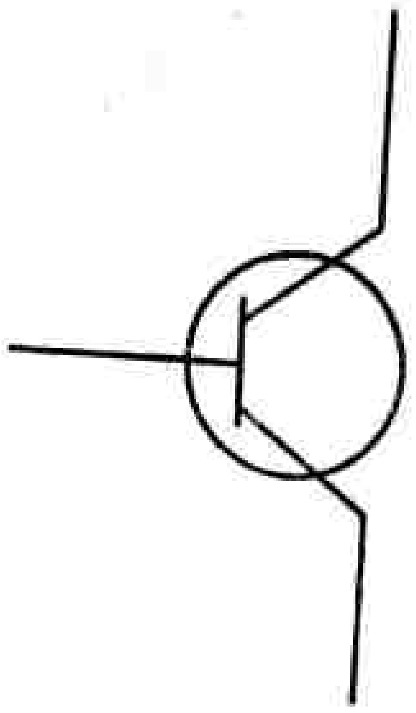


Figure 1. A transistor is easily portrayed schematically by a sketch in two space.

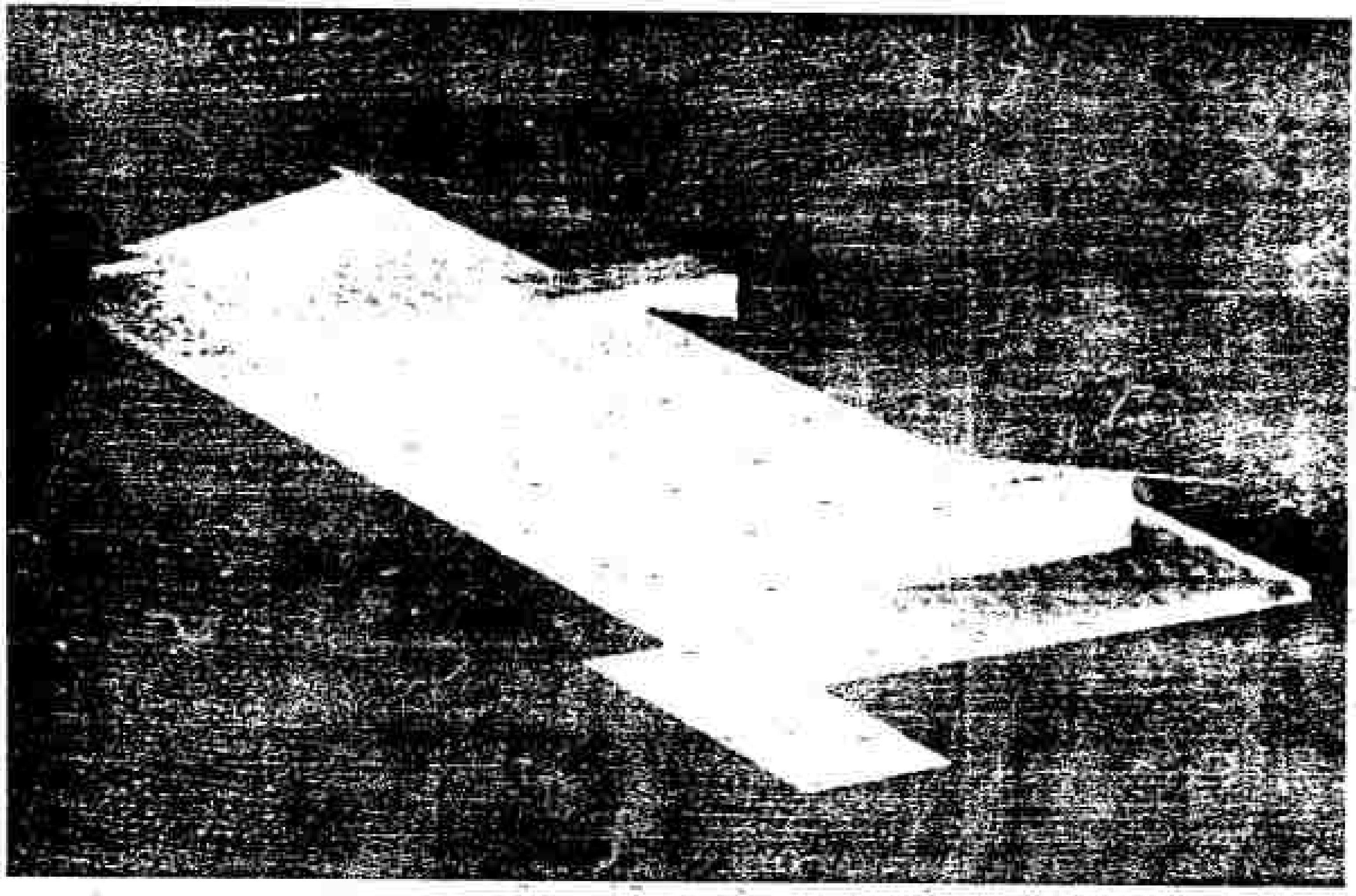


Figure 2. Simple object created interactively at the display console.

Section 1

BASIC CONCEPTS

Objects and Envelopes

Two fundamental modeling entities are objects and envelopes.

Objects represent physical things -- a block of concrete, or a plate of glass, for example.

Envelopes define regions in three-space a opposed to material objects.

Envelopes are analogous to political boundaries. The States are wholly contained in the United States An object in a state is in both regions at once, not one or the other.

The "grain" or fineness of the model is reflected by the number of levels of nested envelopes. If the model is to represent cities within states and building within cities, additional branches to smaller envelopes c n be added (figures 3 and 4).

Objects are abstract representations of real matter. As such, all the properties of matter may be specif ed, such as color, mass, finish, boundary surfaces etc

Real objects cannot interpenetrate and neither should their abstract representations. One object must be ut away to make room for another one.

A basic property of objects and envelopes is th ir geometric shape. The most common shape is the cuboi The cylinder and a triangular wedge are useful also In

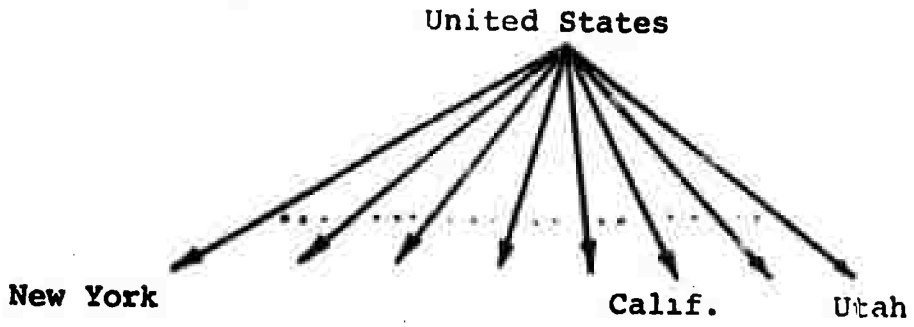


Figure 3. Model of the United States showing the division into states.

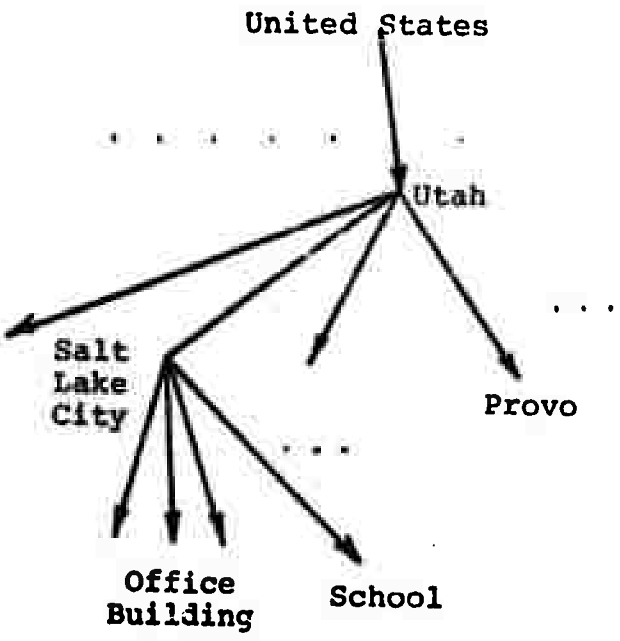


Figure 4. Model of the United States at a finer grain showing the divisions and objects within a state.

addition, the boundary surfaces of envelopes and objects can be warped or otherwise deformed into complex shapes.

Architects and mechanical designers shape and place both envelopes and objects. A computer-aided design system for their use must deal with both types of entities.

To better understand how the two interact, consider the model of a room. A room is a region of space and, hence, an envelope in the model (figure 5). Rooms do not exist in the way steel beams and concrete do. Rather, they are spaces which exist first in the designer's mind, and then because walls enclose them.

"Walls" are envelopes or regions in which actual building materials may be placed. The room envelope and wall envelopes encompass or delimit some of the same space. They are not thought of as interpenetrating, however.

A window region is defined in a wall envelope by simply placing the window at the desired spot. To put a window object (solid, such as glass) in a wall defined to be an object, an opening for it must first be cut in the wall. The data structure models distinctly:

(1) the wall;

(2) an opening in the wall; and

(3) a window assembly positioned in that opening.

Trees and Graphs

To be useful, a model must have structure. Components of the model such as rooms are themselves composed of com-

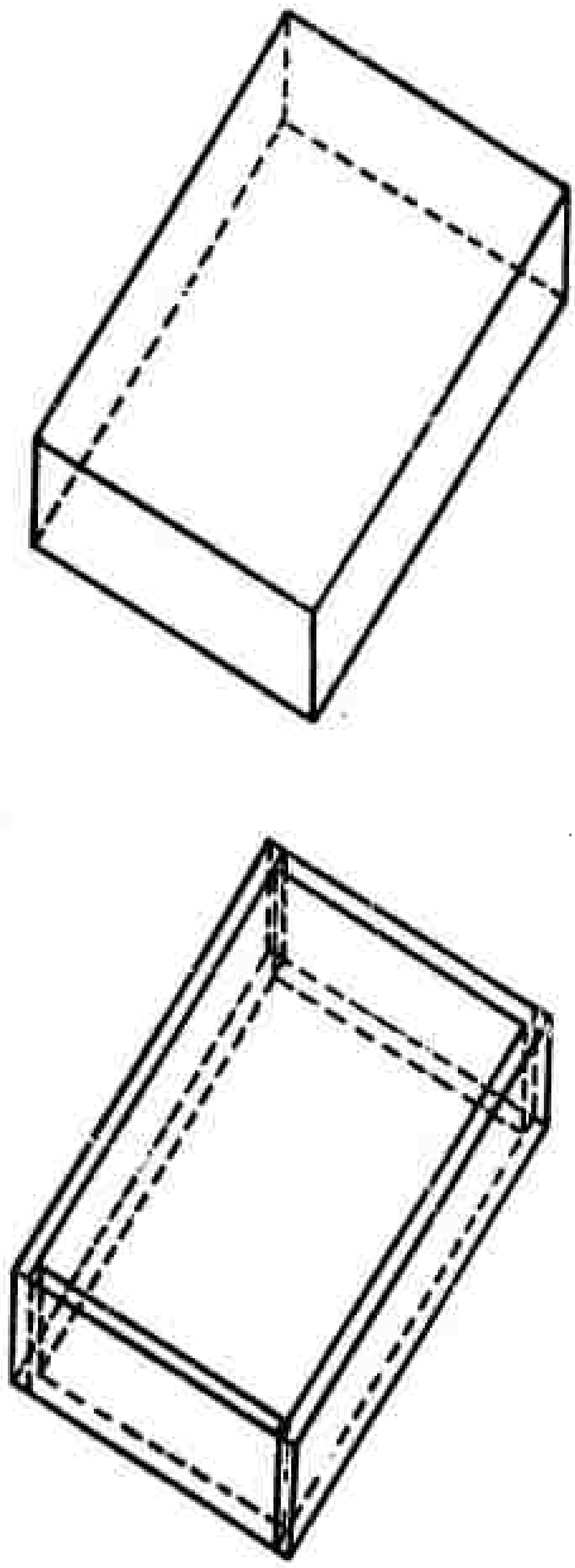


Figure 5. Room envelope containing wall envelopes.

ponents (e.g. walls, doors, etc.). Such heirarchies can be modeled by mathematical graphs namely collections of nodes and branches. A particularly useful class of graphs are trees (figures 3 and 4 show two examples). Techniques exist for storing and manipulating trees within the computer. These details are discussed briefly in Appendix II.

Walking the tree refers to algorithms which select particular nodes according to some sequence in order to process the tree in some way. Trees can be traversed in many ways. The following illustrates only one possible way of accomplishing this traversal.

To display a tree for example, the tree walking procedure beings at the top node and advances down the left most branch repeatedly until a terminal node (leaf) is encountered. Next the algorithm walks back up one level over one branch to the right and then down again, etc.

Copies and Instances

As explained previously, a tree structure represents something which has been designed. This might be a window or a chair in an architectural model. The structure is given a name and can be referenced by other structures in two uniquely different ways. One way is to copy the entire tree over into the structure where it is needed (i.e., copy), giving it a new name in the process. The other method is simply to reference the object by name (i.e., instance). In computer programming terms, the copy is a macro; the instance is a closed subroutine.

Consider a wall with two windows of Type X placed in it. If the windows are in the wall as instances, then the modeling structure is as shown in Figure 6. If the windows are referenced as copies, the model is as shown in Figure 7. The two pictures look the same, but the model is quite different and has different properties.

In the first situation (instance) a change in the design of the window is reflected in both windows in the picture since both windows are referenced as instances. In the second situation, each copy acts independently. A change to window number one leaves window number two unaltered. When modeling a high rise building with 1,000 windows of five different types, the proper choice between instances and copies is critically important.

Making Copies of Instances

The window in the previous section is most likely an entire tree structure, rather than a single node (figure 8). Making a copy of the window amounts to copying its entire structure and assigning this copy a new name (figure 9). Since all the nodes in the model of the window have been copied, and every node must have a unique name, new names are generated for each copied node. The copy operation applied to a complex object generates considerable additional data, which must be stored somewhere. This is a practical reason to restrict the use of copies.

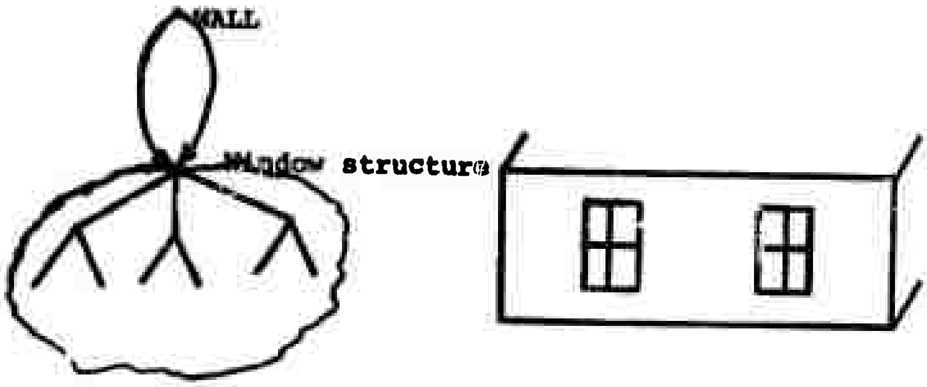


Figure 6. Windows referenced as instances.

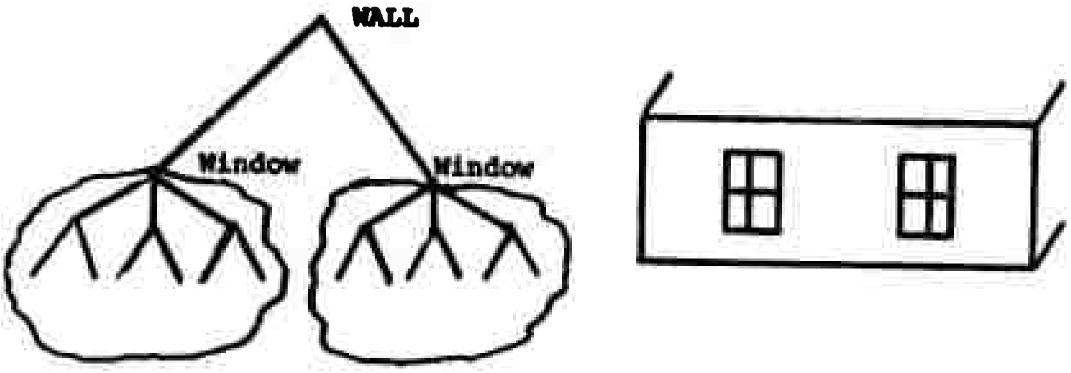


Figure 7. Windows referenced as independent copies.

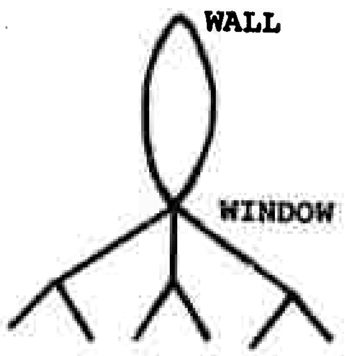


Figure 8. Two instances of a tree structure defining a window are referenced in a wall.

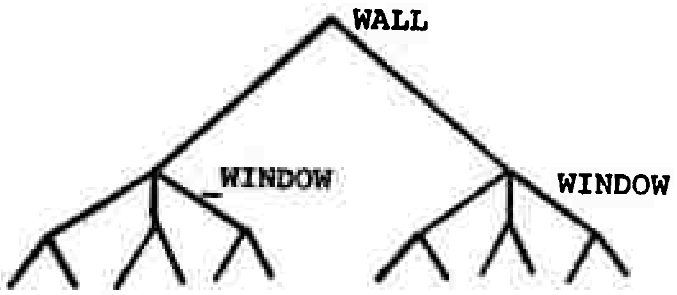


Figure 9. Two copies of a tree structure defining a window are referenced in a wall.

Constraint Declaration and Satisfaction

Constraints are declarations of relationships between two or more objects. For example, the plumbing in a building should be constrained to stay in the walls, even as the walls are moved about and repositioned. Powerful computer-aided design systems should provide the user with some way of declaring constraints and then solving them or, at least, signalling when a constraint is violated.

Before this processing is done, constraints specified by the user are usually reformulated by the computer, rendering them more manipulatible.

General Constraint Satisfaction

Given a set of relationships, the goal of constraint satifaction is to choose values of the variables which result in all constraint relationships being satisfied. Typically, there are more vaiables than constraints, hence infinitely many possible solutions exist. For example, the constraint that two walls remain parallel admits to infinitely many orientations of the walls in a floor plane.

General constraint satisfaction is a task akin to theorem proving. It is a hard problem that is as yet unsolved.

Constraint Representation for Geometric Modeling

For many situations in Geometric Modeling, the solving of arbitrary constraints is not necessary. The choice of representation can facilitate constraint processing or provide an implicit constraint solution, in a number of cases. For example, a house might be defined as a set of rooms positioned in some manner. The rooms in turn could be defined as a set of four wall envelopes (figure 10). By this definition, the walls move with their respective rooms when the configuration of rooms is changed.

An alternative definition is a house composed of a set of rooms and walls (figure 11). By this definition, the rooms and walls move independently, and explicit constraint declarations are needed to keep the walls with the rooms as the rooms are moved.

Another essential representation decision for constraint satisfaction is the property of proper inclusion. An additional piece of data at each node describes an envelope, which is the smallest cuboid that completely encloses all the components of the object. These envelopes allow an algorithm to quickly determine if two ojects attached to different tree structures are accidently interpenetrating. The proper inclusion requirement turns out to be a very natural one where buildings are concerned. The designer intuitively thinks of the walls as parts of a house, and not the other way around. The system permits him to structure an object in the reverse order, should he want to do so, in which case interpenetration checker simply runs more slowly.

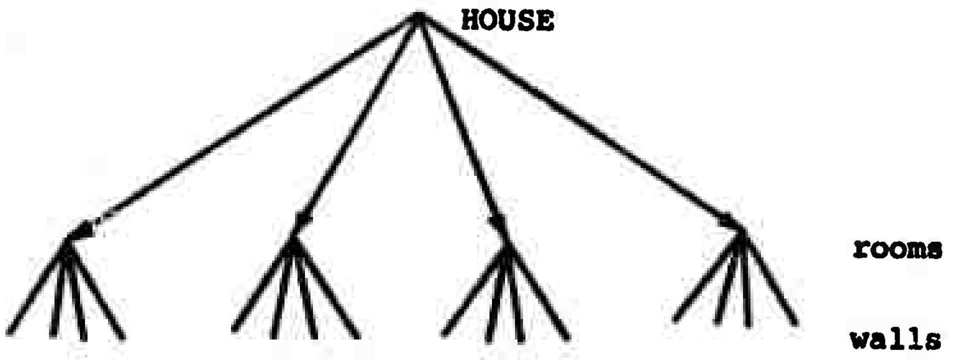


Figure 10. Definition to avoid constraint processing.

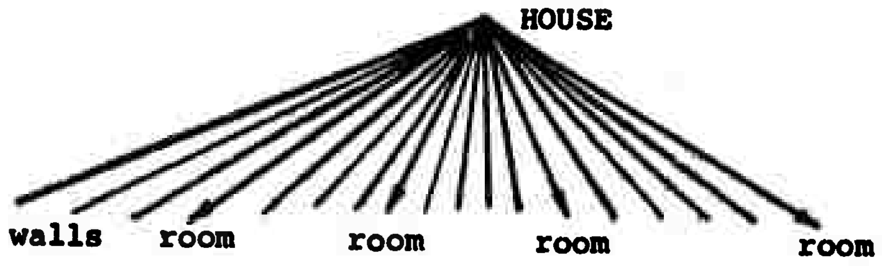


Figure 11. Definition of HOUSE which requires a constraint processor to maintain the walls with rooms.

Modeling Elements - Line, Surfaces and Volumes

Suppose we wish to represent a solid pentagon (figure 12). A line representation of this object has the x, y, z coordinates for the end points of each edge of the object as separate data blocks. A surface representation contains seven data blocks, each containing the x, y, z coordinates of the end points of the boundary lines stored in an appropriate order. A volume representation amounts to one block containing the coordinates of the extreme points in an order to indicate how they relate to one another.

A line representation of an object requires the most storage space and a volume representation the least. Both extremes are needed at various times. However, an object stored as a volume can be easily converted to a set of surfaces. Given a set of surfaces, it is extremely difficult to deduce what volumetric shape they represent (cube, prism, etc.), if indeed they do at all. One direction is simple and algorithmic; the other is complex and probably heuristic. The physical analog of this is a ball on a ledge. With very little work (corresponding to computing time), the ball can be knocked off the ledge to reach the other state. Considerable work (computing) is required to raise the ball to the ledge from the plateau below (figure 14).

If the task is to determine which surfaces are hidden from a particular vantage point and which are visible, then data can reasonably be stored as a set of surfaces

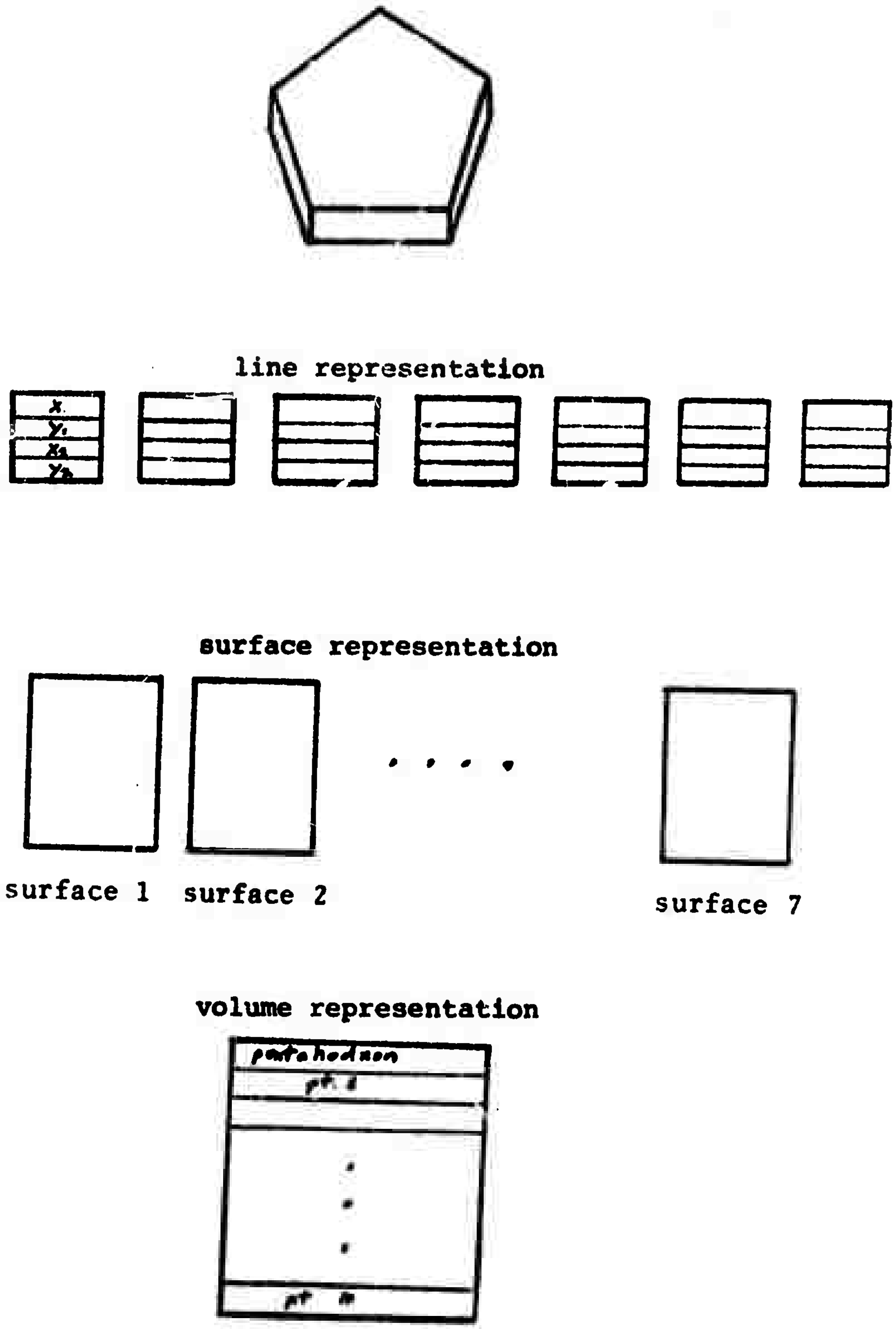


Figure 12. **Various computer representations of a pentahedron.**

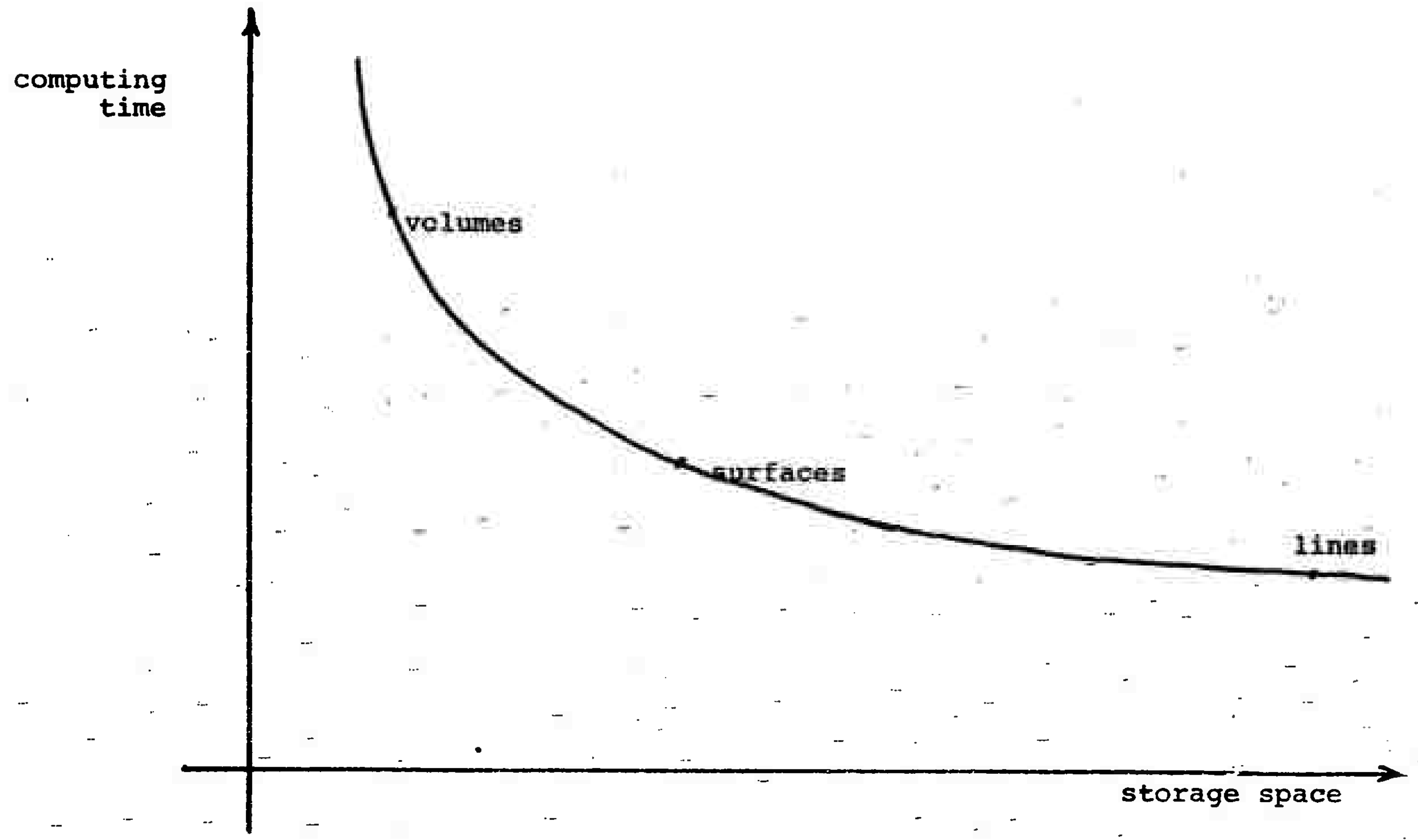


Figure 13. The trade-off between computing time and storage space for objects which are ultimately used as lines.

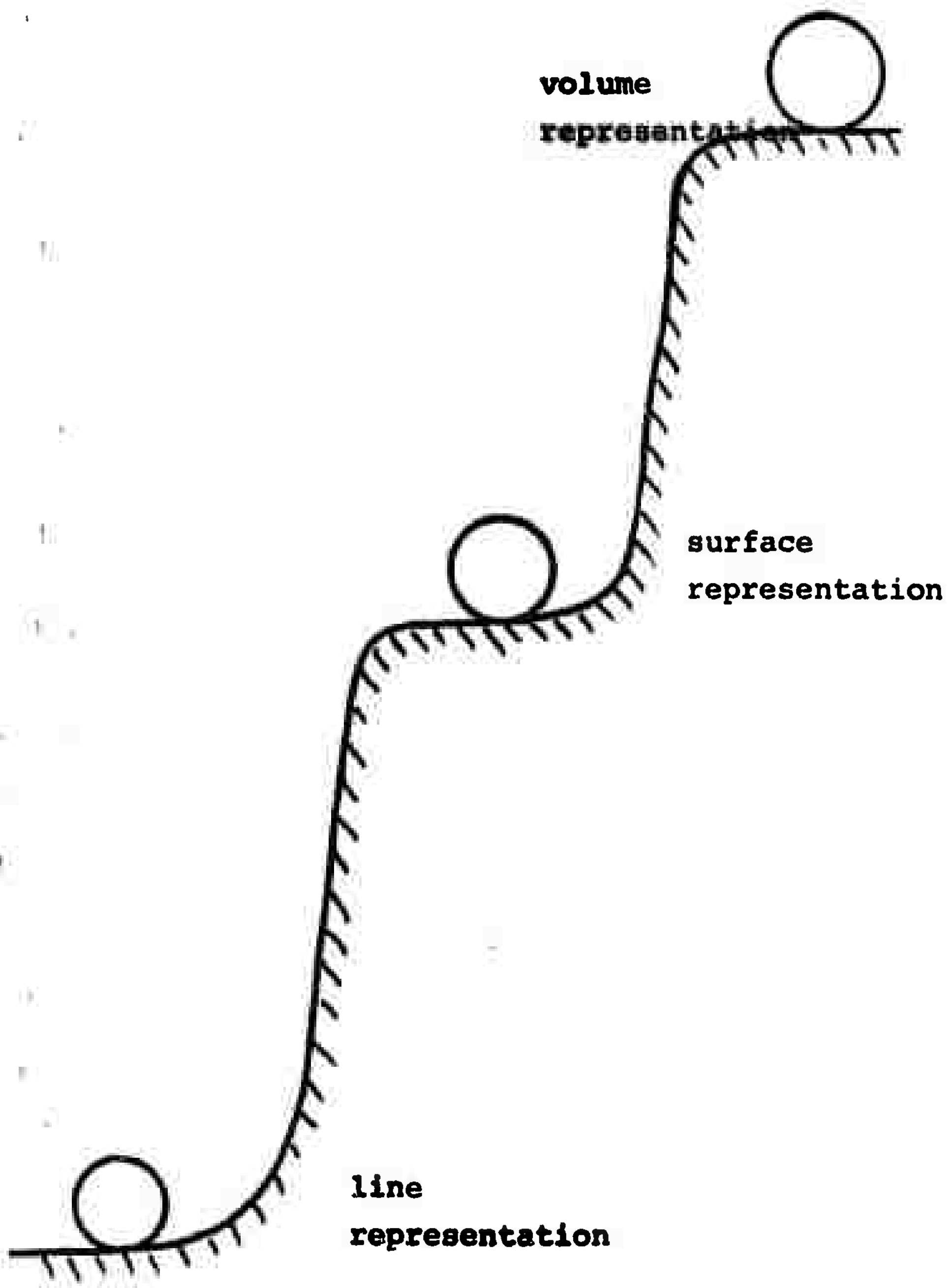


Figure 14. **Analogy between work to change the state of a physical object and the computing work required to change the state of information representation.**

with associated normal vectors [10]. For most other processing, however, a surface representation is cumbersome. Suppose the volume (or weight given a material) of the parallelepipeds is to be determined. Stored as a volume, this computation is trivial while, stored as a set of surfaces, it is difficult, since the surfaces must be checked pairwise for common edges. Recovering the data as volume is a difficult and lengthy task.

Geometric modeling is usually done as a part of an interactive design system. To be economically feasible, such systems must do more than simply present and manipulate pictures. Facilities must exist to attach arbitrary attributes (such as weight and cost) to objects, and the meaningful elements with which to associate this information are solid objects.

When planes, lines and points are occasionally needed in the model, they are easily represented as degenerate cases of volumes. While it is true that many one and two-dimensional objects appear in the pictures derived from the three-dimensional model, they are not particularly useful as elements of the model itself, however. A plane is frequently used for sectioning an object for viewing purposes. It is important to recognize, however, that this plane is the result of the sectioning algorithm and is <u>not</u> part of the data of the object itself.

Section 2

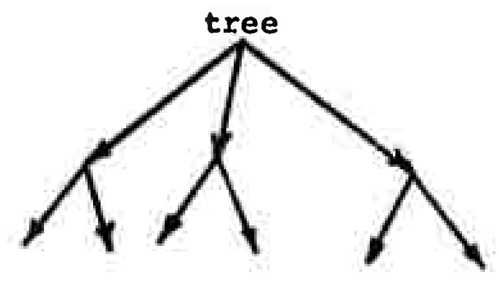
THE MODEL

The Model

The basic modeling structure proposed here is a type of tree called a semi-tree (figure 15). Trees have two properties: (1) a single root node exists from which the tree grows; and (2) a unique path exists from the root node to any leaf. Semi-trees satisfy the first property, but not the second. Both types of trees are free of loops, but more than one path can exist from the root of a semi-tree to a leaf.

The semi-tree structure is not new. It is essentially the instance-and-copy structure used for modeling in the past. The problem is how to model a building with such a structure so that manipulations on the building happen smoothly and naturally.

Consider the model of a house. It consists of rooms and the rooms are composed of walls, a floor, and a ceiling. (figure 10). The question arises as to whether the walls should be modeled as a part of the house or a part of the individual rooms. If the walls are elements of the rooms, then two rooms have a wall in common. Another possibility is to have one wall envelope of each room occupy the common volume between the rooms (figure 16). Walls in the model shown are represented as envelopes, and therefore do not interpenetrate each other. They simply delineate regions in the rooms. On a display screen this appears in plan view as is



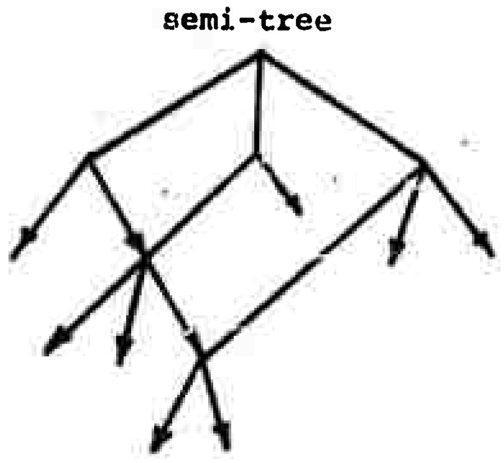


Figure 15. Two important special classes of graphs.

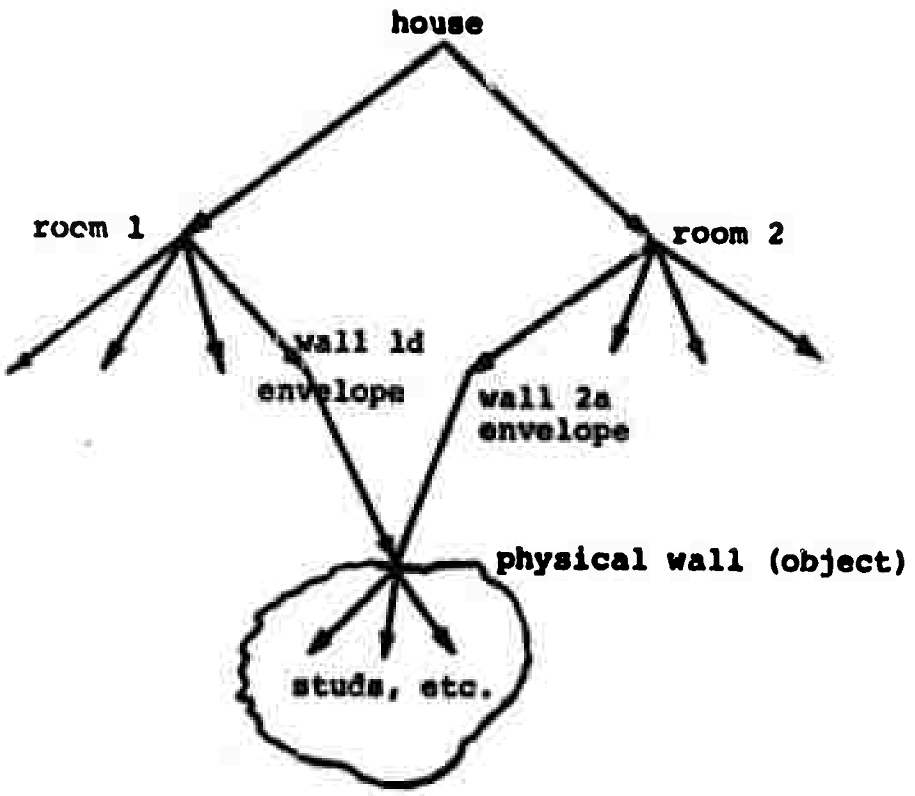


Figure 16. Use of envelopes to define the region where a physical wall object will be placed.

shown in figure 17.

What should happen if one room is removed? Is an intermediate wall associated with one room or the other or directly with the house? What will happen depends on the way the wall envelopes are assigned to nodes and the way these nodes are referenced.

This author has found the structure described above to be fairly natural for representing structures such as rooms designed separately which must fit together in a building with common walls. Each room is designed with the walls around it defined as envelopes or regions. After the rooms are in place, wall material is positioned in the envelope region between rooms (figure 16).

Each node of the semi-tree has a unique name, such as WINDOW21. Each node in the tree is the root of a sub-tree by that name. Corresponding to the tree structure is a heirarchy of coordinate systems, one for each node of the tree (figure 18). The coordinate axes associated with the root node are inertial axes with no frame of reference. The coordinate systems of immediately dependent nodes are referenced to the coordinates of their dominant nodes.

The relationship of the dependent coordinate system to the dominant one is established by nine parameters: three translations (T), three rotations (R), and three scaling (S) numbers, corresponding to the three orthogonal axes (TRS parameters, for short) (figure 19). Scale factors other

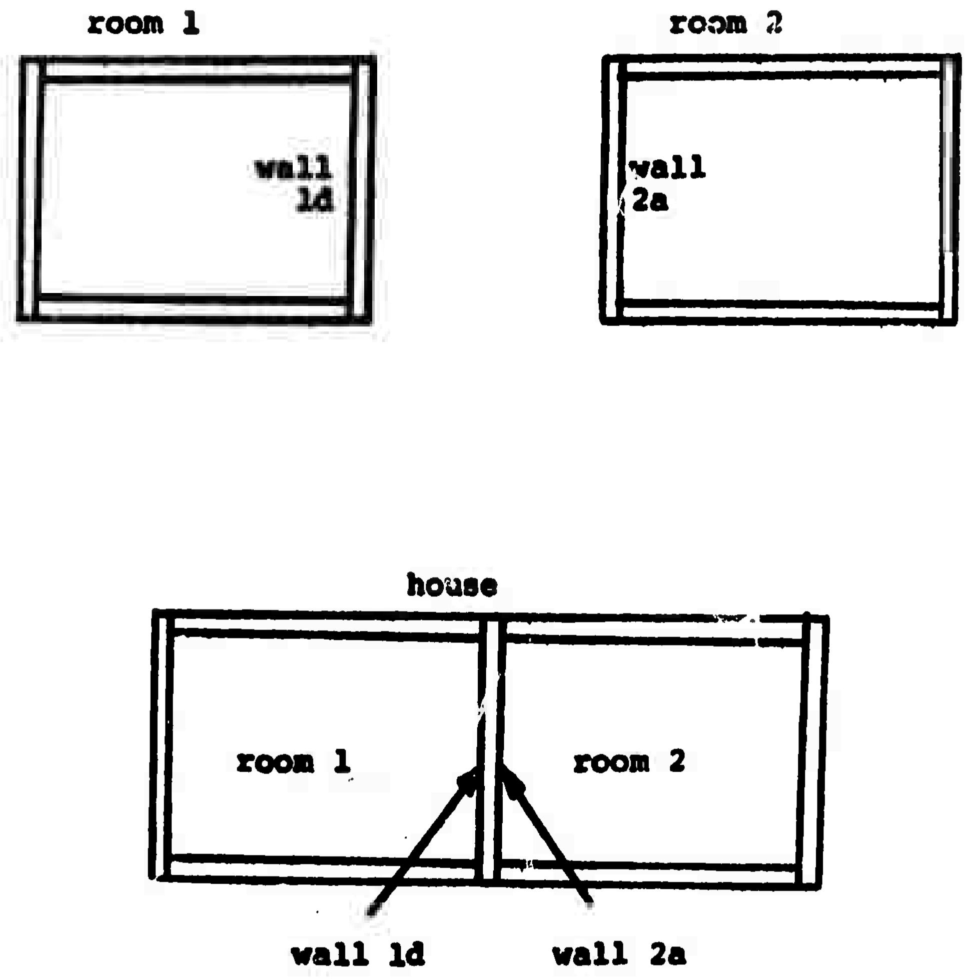


Figure17.Display view corresponding to Figure 16.

$$\begin{tikzcd}
 & & A \arrow[dll] \arrow[dr] & & \\
B \arrow[d] & & & C \arrow[dl] \arrow[d] \arrow[dr] & \\
D & & E & F & G
\end{tikzcd}$$

1. Coo·dinate systems B and C related to system A.
2. Coordinate axes E, F and G are related to C.

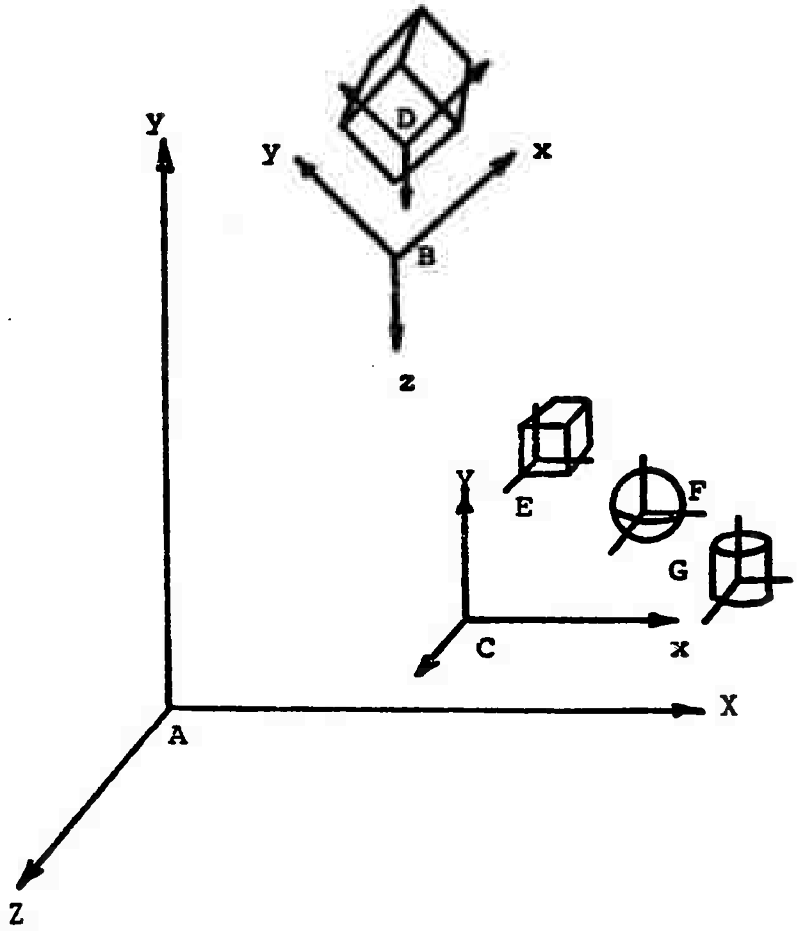


Figure 18. Correspondence between tree stru ture and the coordinate system at each node.

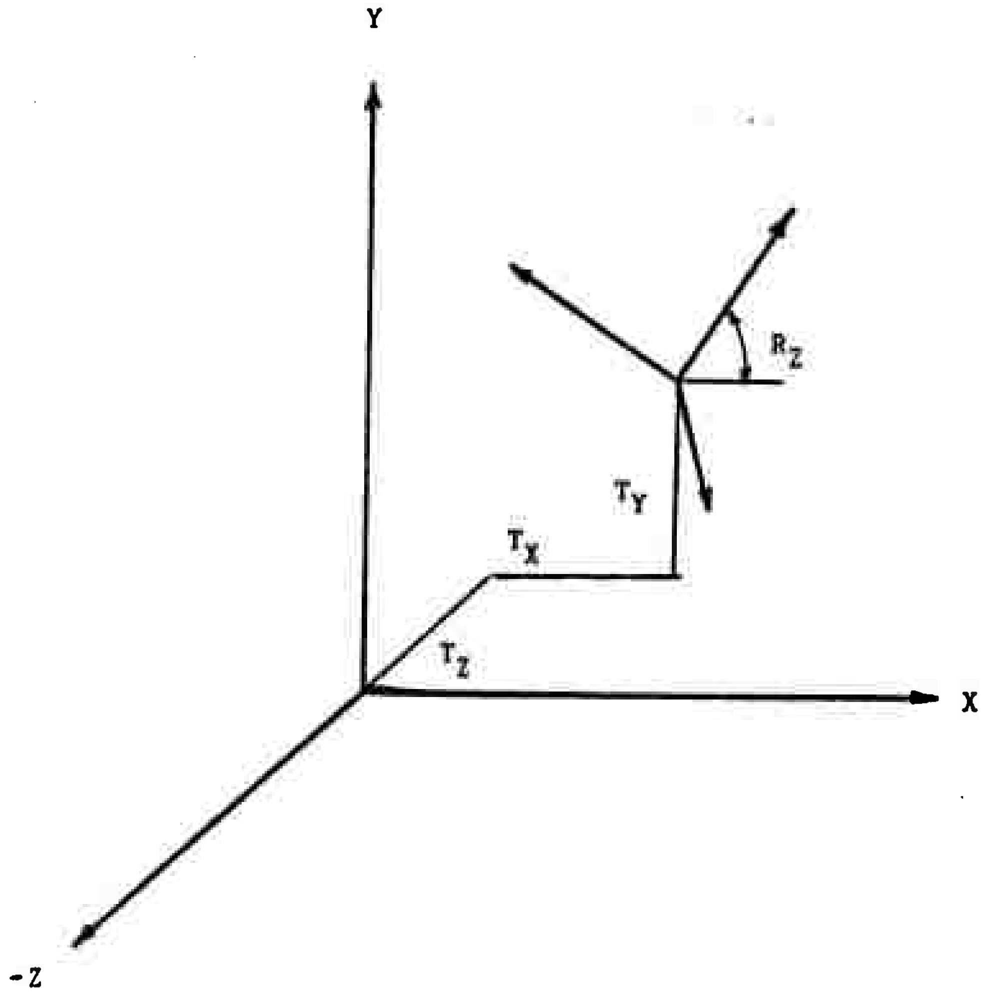


Figure 19. Translation, rotation and scaling parameters expressed in terms of the dominant node's coordinate system relate the dependent node to it.

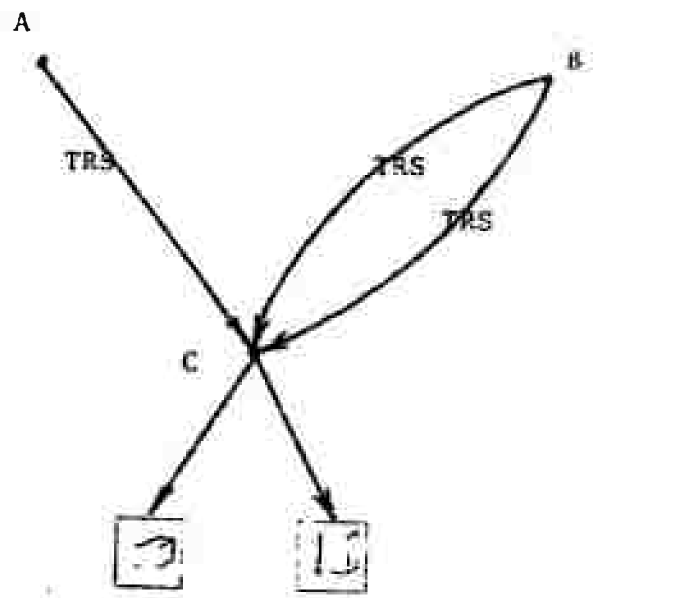
than unity result in magnification or contraction of all objects dependent to that branch.

The translation, rotation and scaling (TRS) parameters are associated with branches, not nodes. One dependent node can be referenced from several places or many times from one place (several instances). The placement of each instance in relation to the coordinate systems of the dominant systems is indicated by the TRS parameters on the branch to that instance (figure 20).

Another qualifier which can appear on any branch is that of subtraction. Subtraction causes the dependent object to be subtracted from the object to which it is attached. The system checks to be sure that undefined or otherwise meaningless situations are not set up in the structure. The property of subtraction is an expression of a suggestion by D. Cohen* on a technique devised by Euler. The bounding edges of a surface are traversed clockwise for exterior edges, counterclockwise for interior edges while facing a point somewhere in the plane of the object. If the sum of the angles is 360, the point is in the object; if zero the point is outside.

Producing a displayable picture of an object represented heirarchically amounts to walking the tree in a post-fix fashion, updating a rotation matrix with each

*Harvard University graduate student in computer science.



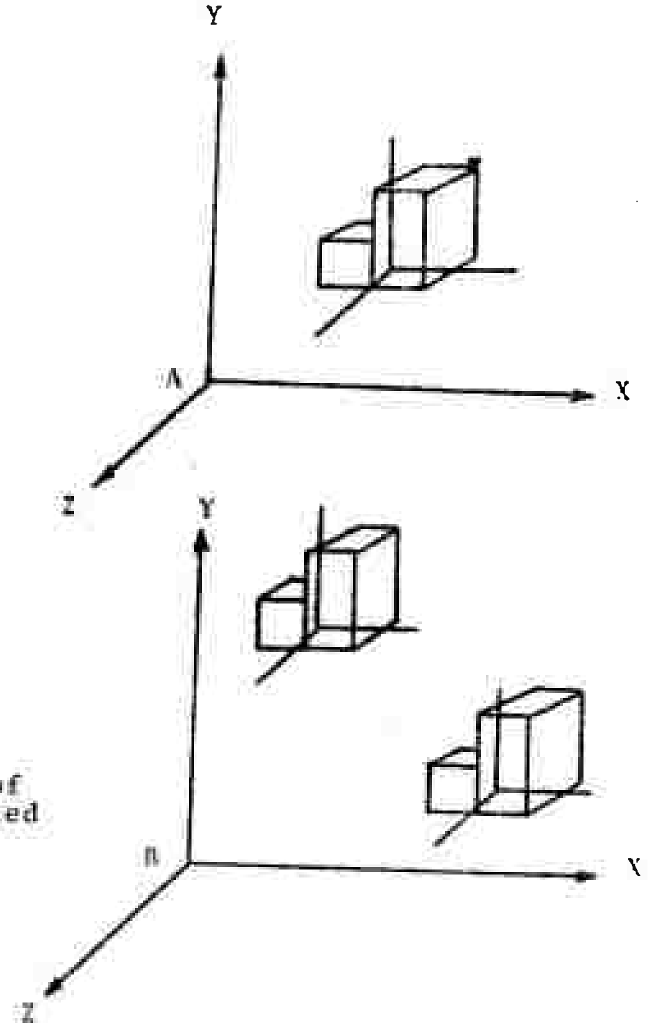


Figure 20. Three-dimensional placement of dependent objects is controlled by the TRS parameter of the branch to the object.

step. When a leaf is encountered, th matrix reflects its rotation translation and scaling appropriate for that leaf.

An alternative scheme is to associate a rotation matrix w.th each object, instead of developing it on the fly. Such an approach involves less computing, but it i: unacceptable, since one object may be referenced as a part of many objects. With each reference, its orienta:ion and, therefore, associated matrix, will be different.

The terminal nodes of the semi-trees are marke' as such and contain data describ.ng basic geometric forms. Typical basic forms are the cuboid, sphere, cylinder, etc. In addition, these basic forms may be stretched, the sides of the cuboid may be warped into a Coons surface in a manner similar to that described by Forrest[8]. In this case, the primitive block holds functional descr:p'io's of the warped surfaces. The edges of the basic geometric object provide boundary conditions for the mathematical descriptions. Warped surfaces are used only infrequently in present-day architecture. Possibly they will be used more often as better tools for designing and building them become available.

In most respects the primitive geometric forms attached to terminal nodes are processed in the same fashion as entire trees. One set of rules applies to all. The primitive forms are mainly used to build up more useful, but still primitive, forms. An I-beam, for example, can be built from three stretched cuboids (figure 21). After it is once defined, it becomes a primitive to a structural designer and can be rotated, stretched and referenced in many places.

Arbitrary attributes can be attached to any node. Attributes typical for computer-aided design are weight, cost, supplier, etc. These data are of the form:

ATTRIBUTE of OBJECT is VALUE.

For example, a node defining a beam might have these relationships attached to it:

NAME OF BEAM is WF30.

WEIGHT OF BEAM is 570.

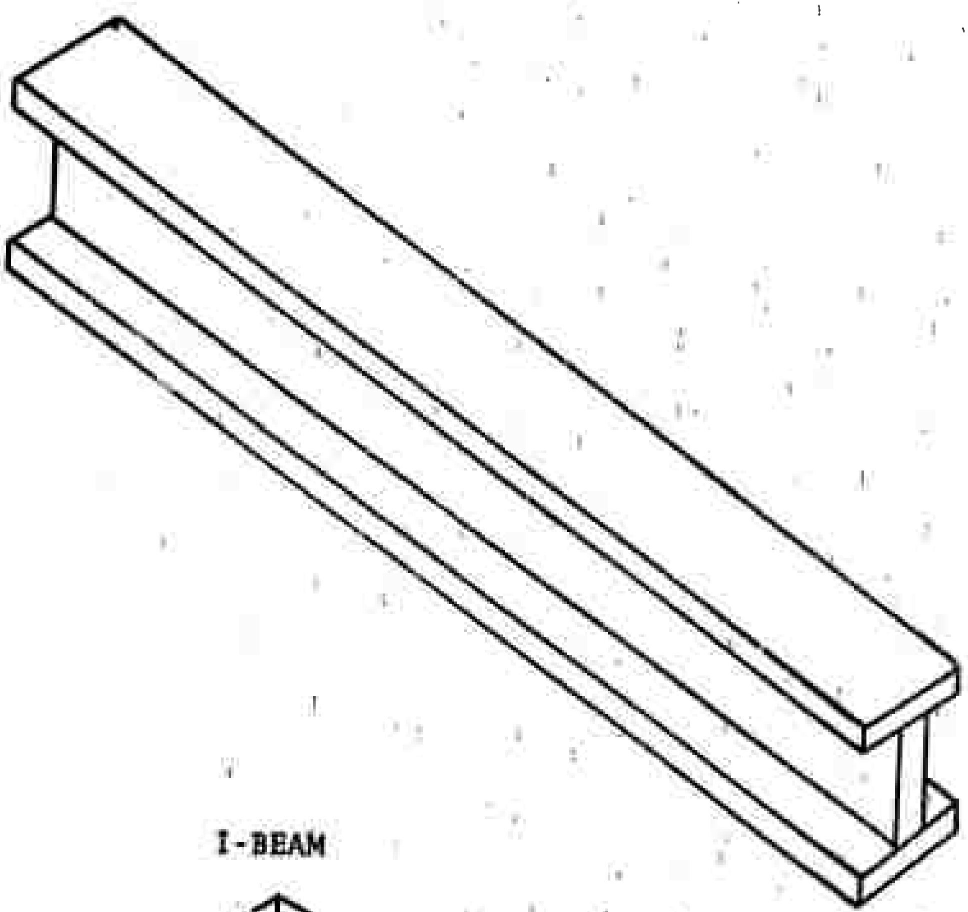
It is essential that the users be free to assign arbitrary attributes to any node in the model. Useful processing programs depend on the presence of descriptive attributes. An example of this occurred recently when it was desired to add color to objects. A description for each color was typed in.

'DESC' * 'RED' = 00075

Subsequently, the user typed:

'COLOR' * 'PART21' = 'RED'.

The only conventional computer programming as such amounted to a procedure in the display routine to check for the presence of a color, look up its description, and pass it on the shaded picture routine. As another example, the smallest enclosing envelopes referred to earlier are actually attached to modes as attributes.



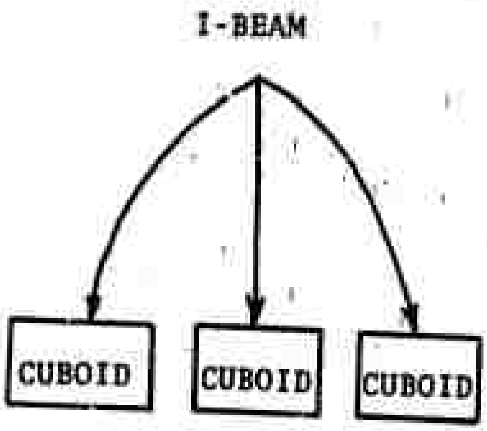


Figure 21. I-beam "primitive" modeled as three stretched cuboids. The system treats this user-defined "primitive" exactly the same as any built-in basic form.

In the building trades, hardware fixtures are represented in a catalog. Designers refer to these basic designs without modifying them (figure 22). This corresponds to subroutining in computer programming. The ability to model this process is essential, since a designer usually begins with standard components. The resulting sturcture, while still tree-like, is even less of a pure tree (figure 22).

Why the Model Works

The viability of the modeling scheme rests with: (1) the ease with which a model is constructed and altered; (2) the faithfulness of the representation; and (3) the ease with which algorithms can manipulate the structure.

The use of user-defined node names and user-specified attributes gives the model considerable flexibility. The ability to reference any previously designed object as part of some new design allows the designer to proceed in a modular and orderly fashion.

The grain or fineness of the model is readily controllable. Greater detail is attained by attaching an entire tree in place of a terminal leaf on an existing coarse tree.

Since there are so few basic elements to the structure, primitive routines define and delete nodes; make and break branches; and attach and remove attributes of nodes. The physical implementation used by the author is a variant

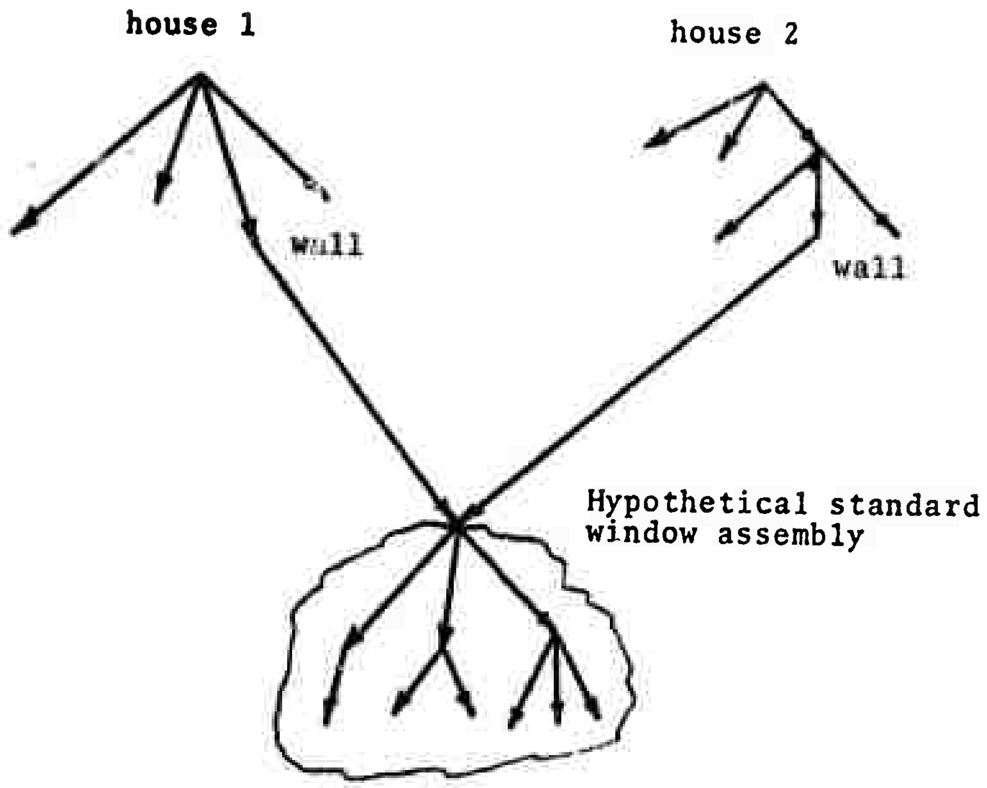


Figure 22. Most large objects are composed primarily of instances of standard things.

of the associative triples scheme of Rovner and Feldman[7]. The use of hashed names instead of pointers allows the structure to work effectively in primary, secondary and tertiary storage.

The faithfulness of the representation depends on the actions of the user in building up a structure. If the user is a typical architect, he will need considerable guidance and experience to become fully effective. He need not become a computer programmer to become facile with the structure, but the considerations to which a programmer is accustomed are germane.

An architect, experienced in the use of a computer as a design tool, might approach the design of a high rise building as follows.

He probably has a broad outline of the building in mind before sitting down at the console. If not, he can stretch and place a few geometric forms, representing the outline of the edifice in general terms.

Next, he considers the functions of different floors. Intermediate floors often have a common general plan. The designer will develop a typical floor plan and then reference instances of this prototype, as required. The few standard office plans can be worked out and referenced as instances with varying orientations and positions in the typical floor plan.

The above description might suggest the model is oriented to the design of modular building, but this is not the case. In the building example, an equipment closet or stairwell could be placed on a particular floor, even if the floor were otherwise an instance of the standard floor plan prototype.

The Partial Copy:

A considerable fraction of any building consists of subassembles, designed by various designers and obtained as standard units. The model of an entire window assembly is best referenced as an instance, since the designer can only select a new window assembly, not alter a part of it.

Partially copying an object amounts to working down each branch of the modeling tree until a node is reached which is flagged as an "instance only" node. Such flags are easily set and changed by attaching them as attributes of nodes.

The representation of a house may require 100,000 words of memory. A copy of it would require the same amount of memory again and this a practical reason for not making full copies.

Algorithm for Quickly Detecting the Interpenetration of Objects

Interpenetration of two objects in the model represents a departure from the real world being modeled, since two physical objects cannot occupy the same space simultaneously. Violations of this kind can easily occur, however. Two structures, designed separately, are moved into close proximity with one another. This is done by orienting the top node of each tree with respect to a dominant third node. Objects attached to the leaves of the trees may interpenetrate as a result. A typical structure has several hundred objects attached. Comparing them individually with a similar number of items attached to another tree woul take far too long (many minutes, in some cases). The user needs to be informed of such difficulties at once so that he can take corrective action. The problem is to do this computation quickly.

The Importance of a Metric

When a large number of objects must be compared to determine if they together exhibit some relationship (e.g., interpenetration), a straightforward approach of pairwise comparison is inadequate. What is needed is some way of determining if two objects are in some sense "close" and therefore must be compared. Such a measure is called a metric.

The metric is simply a criterion for determining how far two objects are from one another. In a two or three-space environment, candidates for the metric are easily found. The common distance function is one.

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (x_1 - z_2)^2}$$

in N-dimensional space with no immediate physical interpretation, an appropriate metric is harder to come by.

The geometric model advanced here has an envelope attached at each node which defines the maximum extent in $\pm x$, $\pm y$ and $\pm z$ of the elements attached to that node. These envelopes provide global information about the entire sub-tree. With this information, a processor can quickly determine if the sub-tree needs to be checked in finer detail.

The Algorithm

The algorithm for determining interpenetrations quickly consists of recursively testing the envelopes associated with the nodes at each level of the semi-tree to determine if they interpenetrate. If they do not, the substructure of the two nodes need be compared no further, since the proper nesting of the envelopes insures that their components cannot interpenetrate if they themselves do not. This fact greatly reduces the computing time, which increases with the amount of overlap of substructures, rather than with the total complexity of the object. After only a few tests down from the top of the tree, most objects are partitioned off from the envelope under consideration and need not be compared with it.

If the envelopes of two nodes at one level do define a region in common, the semi-tree attached to one of the nodes must be walked to determine which of its components intersect with the other node at the first level. These tests are made with a postfix tree walk. This walk is not exhaustive, as each node is checked to see if the tree below it can be skipped

Growth of Computation with Complexity

A figure of merit for an algorithm of this sort is how rapidly the computation time grows as the complexity of the model increases.

The time required for a simple two-at-a-time comparison of a set of items grows by the square of the items involved The algorithm given above has global information, which allows most of the possible tests to be skipped entirely. The nodes at one lev l in the t ee must be compared, t o by two However, there are relatively few of them (typically, five to ten). If they do not intersect, they are not considered again. If they do have a reg on in common, the sub tree o one must be checked against the envelope of he o her. This process is more nearly lin ar, for if the two nodes have little in common, the sub- ee is quickly dispens d with by examining the global in ormation the envelopes provide.

Updating the Bound ng Envelopes

As useful as the envelopes are for quickly detecting interpenetration, they present an updating prob em. Several heuristic procedures have been found which are quite effective. The procedure employed is as follows:

Whenever a structure is connected to the presen design, the new unit is assumed to be interpenetration free within itself. Using the outer dimensions of the new substructure, the envelope of the node to which it is connected is easily updated.

Whenever the user shifts his context to a totally different object (a command exists for this purpose), the entire tree for this object is checked and updated, as required. These changes of context happen relatively infrequently compared to other operations. A full check of the object reasonably insures that it will be internally interpenetration-free when it is used as a component of something else.

Section 3

THE SYSTEM

Experimenting with interacitve computer graphics on a batch processed 1108 is obviously difficult. However, constructing a full-scale time-sharing system before commencing graphics research is unattractive, also. To allow graphics research to begin quickly, a swapping system was added to Exec II by computer center personal to accomodate two users: a batch stream and a single interactive graphics console. This system allows a single user of the Design System described here to interact at less cost than if he commandered the entire 1108 system (approximately one hundred and fifty dollars per hour instead of six hundred dollars per hour or more).

All program development and debugging was done in batch mode, as swap mode provides for execution of binary programs and nothing more. As of June, 1969, work is underway to move the system to the time-shared PDP-10 recently acquired for computer science research. This new system will be more interactive, less costly, and certainly much more fun to use.

The 1108 version of the Design System is currently operational. It is not as interactive as we believe the PDP-10 version will be, but it does provide tools for creating all aspects of the previously described model. At any point in the construction of a model, a halftone photograph can be created, the mechanical structure of the object can be analyzed, etc.

Starting the System

A deck of about 12 cards is read into the 1004 remote batch terminal causing the 1108 system to read in the design system from the Fastrand drum. All further operations are performed at the graphics console, which consists of the display tube, Sylvania tablet, teletype, rotation ball and zoom stick. Initially, the user is in a mode appropriate for the design of space and form.

Modes

Five modes are presently implemented on the 1108 system (figure 23). These are:

1. Spaceform design
2. Halftone production
3. Electrical design
4. Structural analysis
5. Heating analysis

The basic programming is designed and implemented to support an unlimited number of such modes.

Spaceform Mode

Spaceform mode is the basic mode in which the model described in the previous section is created. Spaceform mode, and the under-pinning systems programming for all modes, has been the central responsibility of the author.

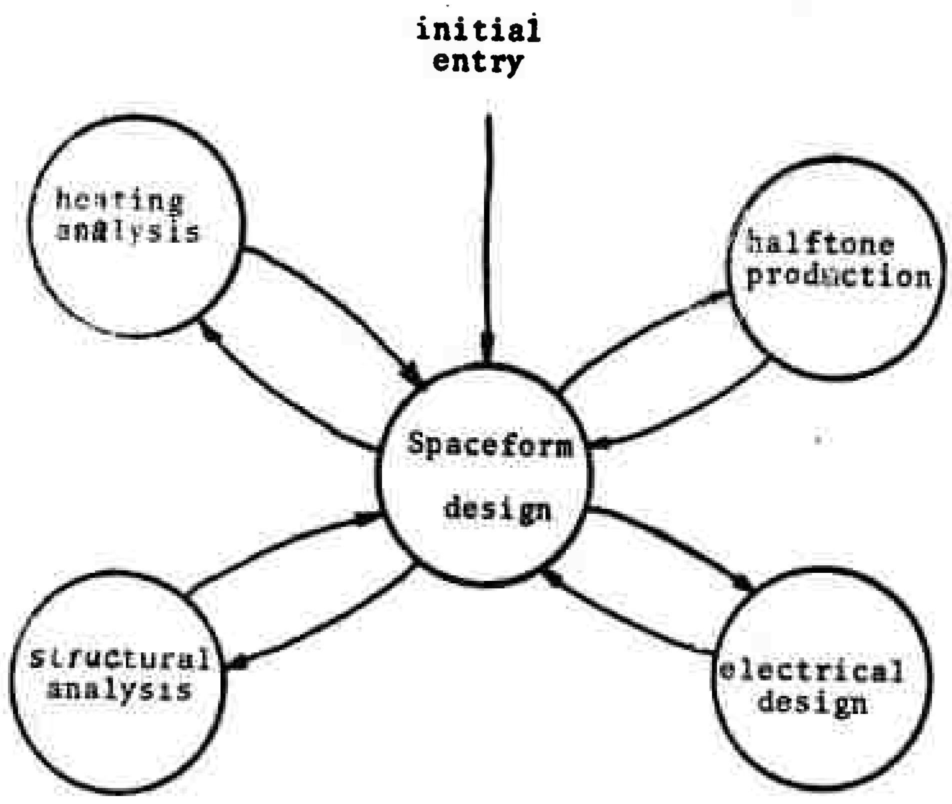


Figure 23. System modes.

Subsystems, such as electrical design, are the responsibility of others and are described elsewhere[5, 12].
In Spaceform mode, an exclamation point is typed by the system to indicate its readiness for user input.

Design Command

DESIGN is often the first command typed by the user.

e.g.: ! DESIGN 'DHWIND'

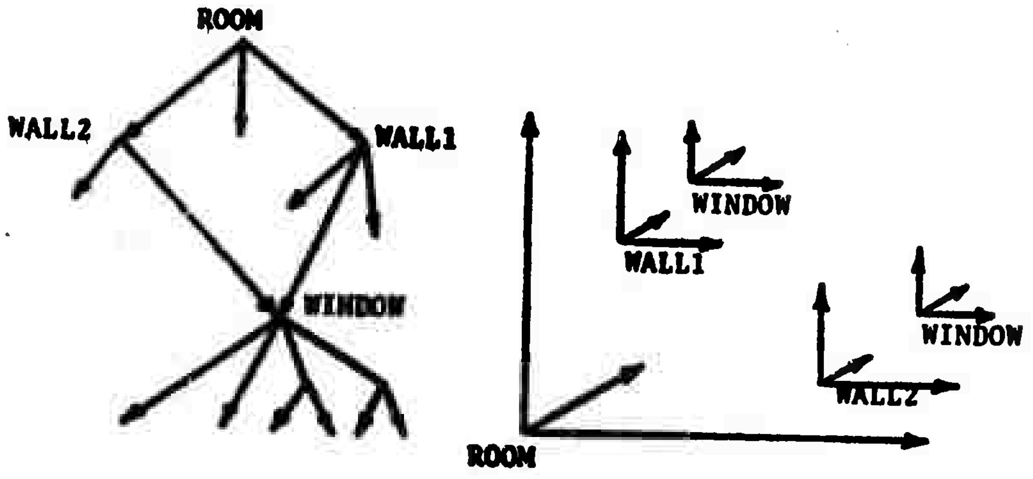
The design command changes the user's context to the indicated object (internally, a modeling tree), which may or may not exist. If the name supplied is not in the data base, a new node is created and the user has a clean slate (internally, a tree of a single node) on which to attach things.

The DESIGN command makes the indicated node the top node for display purposes. Viewing parameters position the structure attached to this node relative to the viewer's inertial frame of reference (figure 24). Commands to be described are available which change the viewing parameters at will without modifying the model of the object in the data base. The context declared with the DESIGN command can be changed by the user at any time.

Primitive Commands

The geometric primitives provided are cube, trinagular rod, hexagonal rod and cylinder.

Design in Data Base



DESIGN 'WINDOW'

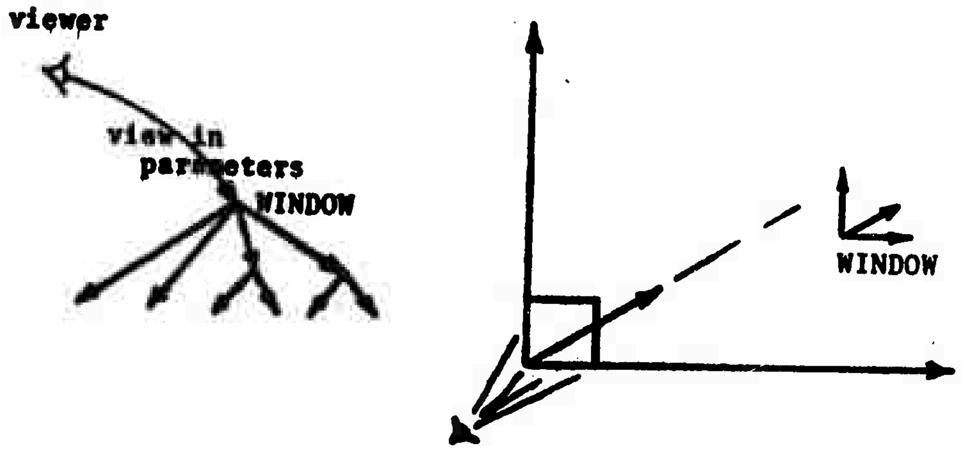


Figure 24. The subtree (object) currently under design is effectively detached from other objects for viewing purposes.

```
!CUBE 'name'
!TRI-ROD 'name'
!HEX-ROD 'name'
!CYLINDER 'name'
```

supplying a name is optional. If the name is absent, the system generates a unique name from a pool of unlikely identifiers. A copy of the indicated primitive is attached to the current top node. The translation, rotation and scaling (TRS) parameters for this branch are set to nominal values which the user will usually want to modify at once with the positional commmands.

<u>Positional Commands</u>

Objects (tree structures) may be translated, rotated or stretched about any or all of the three axes.

```
!TRANSLATE 'A' 20 FEET X
!ROTATE 'B' 10 DEGS Y
!STRETCH 'C' 2 TIMES Z
```

A left-handed coordinate system is used with the viewing window at the origin (figure 25).

If the indicated node (object) is the direct dependent of the top node (object being designed), the TRS parameters associated with its connecting branch are changed accordingly and the object is redisplayed to reflect the change. If, instead, the current top node is indicated, the viewing paramters are altered, changing the way the object appears on the screen, but not altering its representation in the data base.

Stretching an object by naming its top node scales it and all its dependent structure proportionately.

The user is brought closer to the object by typing

!ZOOM IN 10 FEET

or pushing forward on the stick. The reverse operation is ZOOM OUT 10 FEET or pulling the stick towards the user. The idea is that of a viewer flying around a stationary building.

<u>Instance</u>

An instance of a structure is obtained by connecting a branch to it from the curent top node.

INSTANCE 'name'

Nominal TRS parameters are assigned to the branch just as is done for primitives. The position of the instance relative to the axes of the current top node can be changed at will with the previously described commands.

It is the user's responsibility to insure that infinite loops are not established via the INSTANCE command (figure 26). A graph with a loop is no longer a semi-tree. The situation modeled by a loop is so unreal that in practice it is unlikely to occur. In brief, it is an object, one of whose components is the object itself. The system informs the user that such a condition exists as soon as he attempts to display it.

data base.

Stretching an object by naming its top node scales it and all its dependent structure proportionately.

The user is brought closer to the object by typing

!ZOOM IN 10 FEET

or pushing forward on the stick. The reverse operation is ZOOM OUT 10 FEET or pulling the stick towards the user. The idea is that of a viewer flying around a stationary building.

<u>Instance</u>

An instance of a structure is obtained by connecting a branch to it from the curent top node

INSTANCE 'name'

Nominal TRS parameters are assigned to the branch just as is done for primitives. The position of the instance relative to the axes of the current top node can be changed at will with the previously described commands.

It is the user's responsibility to insure that infinite loops are not established via the INSTANCE command (figure 26). A graph with a loop is no longer a semi-tree. The situation modeled by a loop is so unreal that in practice it is unlikely to occur. In brief, it is an object, one of whose components is the object itself. The system informs the user that such a condition exists as soon as he attempts to display it.

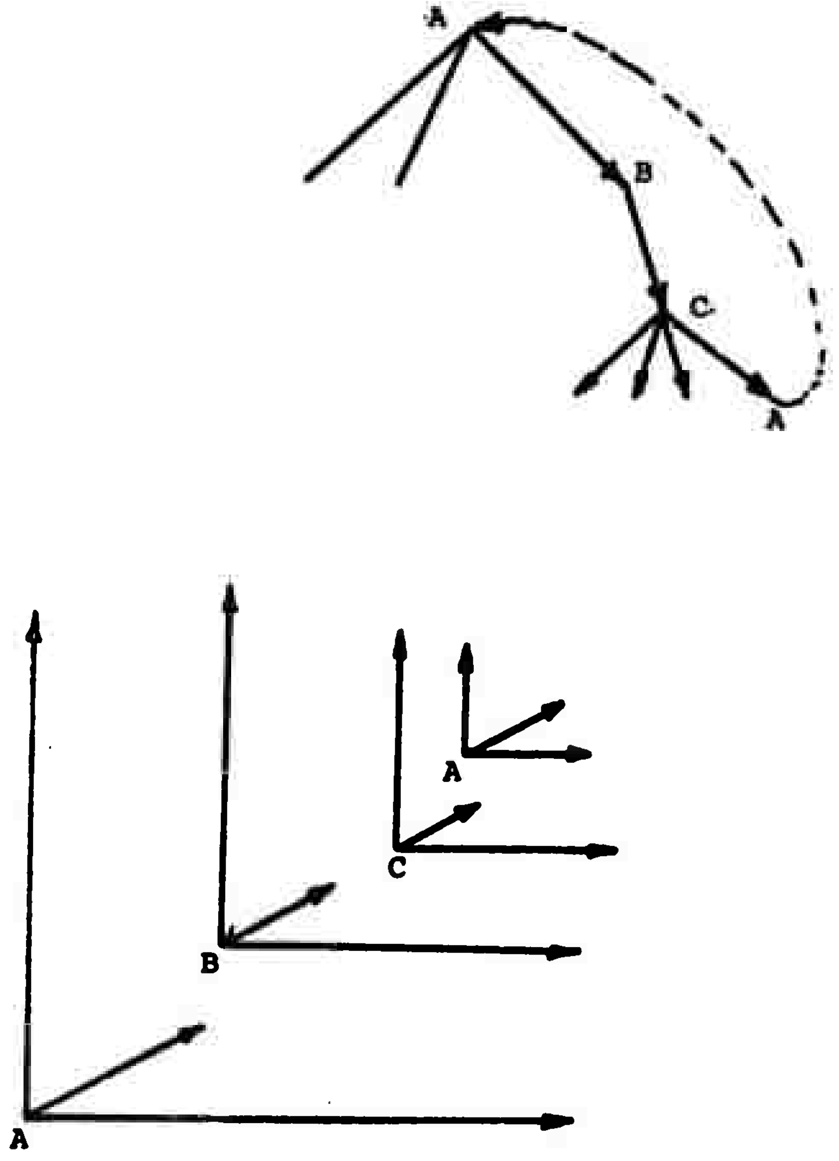


Figure 26. The object 'A' is a component of itself. Unreal!

Copies

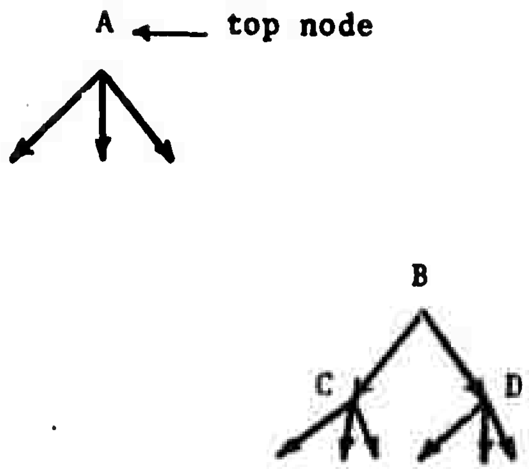
The facility to produce a copy is implemented as a partial copy. That is, a copy is made of the indicated node, and it is assigned a new and unique name and attached to the current top node. The nodes dependent from the copied one are referenced as instances from the new copy (figure 27).

Partial copies are expedient from the point of view of implementation, but they are also what is usually desired by the user. A recursive copy-producing command that is applied repeatedly for many levels, or until some other condition is satisfied, is a possible future extension. A method for automatically choosing all the new unique names would need to be be developed for this to work.

Aids to Interaction

The author does not believe the ultimate system should be as keyboard-oriented as is the 1108 implementation. Button boxes and light buttons are only incrementally better, however, not major improvements. Plans are underway to try some radical methods of direct interaction on the PDP-10 implementation along the lines described in reference 5. The 1108 Exec II swap mode proved too clumsy to make research in highly-interactive graphics worth attempting.

In the interim, a number of minor things have been before



COPY 'B' AS 'BB'

after

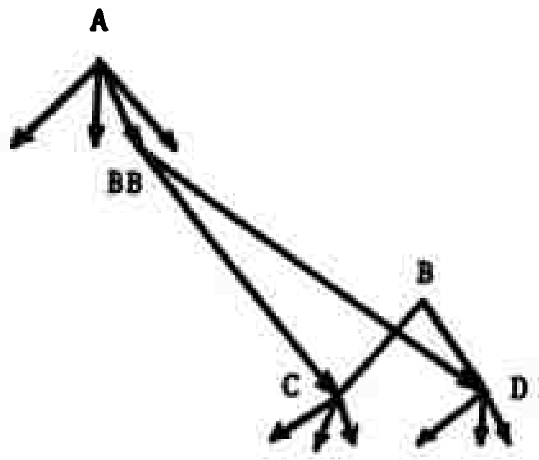


Figure 27. Creation of a copy.

added to the 1108 system to facilitate its use. For one, a node name need be typed only once in a sequence of commands relating to it:

For example:

```
INSTANCE 'B'
TRANSLATE 'B' 10 FEET X
CUBE 'C'
STRETCH 'C' .1 TIMES X
```

can be typed as:

```
INSTANCE 'B'
TRANSLATE 10 FEET X
CUBE 'C'
STRETCH .1 TIMES X
```

Various units of lineal measure are permissible. Initially, distances are in feet. Subsequently, the user types distances in other units and the system makes the conversion. For example, the first time 17 INCHES is given as a distance, the system counters with HOW MANY INCHES PER FOOT?. The user supplies the conversion factor '12.0 , which is stored away for automatic use in the future.

The Sylvania tablet stylus can be used to select an object on the screen, obviating the need to type or even know its name. This allows the user to place primitives without naming them. Future references are made by pointing, which retrieves the system-assigned name.

Saving Designs for Later Use

A terminal session is ended by typing STOP. Before the design system actually signs off, all design-related data in core is written out on the Fastrand for later use. This Fastrand region is 700,000 words and, though large, is too small to hold all designs.

Facilities are provided for writing out tree structures onto magnetic tape and reloading them at a later time. Exec II swap mode (stifling as always) does not allow tape drives to be assigned to the graphics user's program. Therefore, two batch programs are available to be run before and after the terminal session. DUMP OBJECT reads data cards containing the names of the top nodes of tree structures on the Fastrand. The entire trees attached to these nodes are written on tape. Load design performs the converse operation.

Halftone Production

A halftone photograph of the object currently on the screen is produced by typing PHOTOGRAPH. The 1108 computes for about 30 seconds, after which the design system types READY. With the Polaroid camera in readiness, type SEND. The same view can be transmitted any number of times. To return to the Spaceform design mode, type EXIT.

Electrical, Structural, Heating

The electrical design and structural and heating analysis modes are entered by typing

```
    !ELECTRICAL
    !STRUCTURAL
    !HEATING
```

The object under design (current top node) when these commands are given is the geometric object treated by the node.

BIBLIOGRAPHY

1. Carr, C.S. Design System Technical Report. Technical Report 4-16. Salt Lake City, Utah: University of Utah, 1969.

2. Christensen, Carl and Pinson, Elliot N. "Multi-Function Graphics for a Large Computer System," AFIPS Conference Proceedings, Fall Joint Computer Conference. XXXI (1967), 697-712

3. Forrest, A.R. Curves and Surfaces for Computer-Aided Design. England: Joint Computer-Aided Design Group, University of Cambridge, July, 968.

4. Luh, J.Y. S. and Krolak, R.J. "A Mathematical Model for Mechanical Part Description," Communications of ACM. VIII, No 2 (February, 1965), 125-28.

5. Luther, [illegible].A. Electrical Design System. Work in progress. Salt Lake City, Utah: University of Utah, 1969.

6. Roberts, L.G. Machine Perception of Three-Dimensional Solids. Technical Report No. 315. Lexington Massachusetts· Lincoln Laboratories, May, 1963

7. Rovner, Paul D. and Feldman, Jerome A. The Leap Language and Data Structure. Technical Report Lexington, Massachusetts: Lincoln Laboratories October, 1967.

8. Smith, M.J., Macdonald, S.L., and Carr, C.S. Space Form Computer-Aided Design for Architecture. Technical Report 1-2. Salt Lake City, Utah: University of Utah, September, 1968.

9. Sutherland, I.E. SKETCHPAD: A Man-Machine Graphical Communication System. Technical Report No [illegible]6. Lexington, Massachusetts Lincoln Laboratories, January, 1963.

10. Warnock, J.E. A Hidden Line Algorithm for Half-tone Picture Representation. Technical Report 4-5. Salt Lake City, Utah: University of Utah, May, 1968.

11. Wehrli, Robert, Smith, M.J Space-Form II, A Description of ARCAID, the Architects' Automated Interactive Design System. Technical Report 1-3. Salt Lake City, Utah: University of Utah March, 1969.

12. West, Farrin W. <u>Structural Analysis System</u>. Work in progress. Salt Lake City, Utah: University of Utah, 1969.

Appendix I

INTERPENETRATION ALGORITHM

The interpenetration algorithm has been implemented as an independent program on the 1108 but will not be included in the design system on the 1108. It is planned for inclusion with the PDP-10 version, however. Preliminary tests indicate that computing time is directly related to the total number of nodes in the tree and the number of overlapping envelopes. The method advanced here is always faster than the brute force comparison of two nodes at a time. It is at its best with trees that are short and fat, since they admit to the greatest amount of early pruning of entire subtrees.

It is intended that the system will signal the user of a violation by indicating the two objects that are interpenetrating and the dominate objects of which they are a part. This is done when the first violation occurs. The user must correct the situation before proceeding.

Appendix II

SYSTEMS PROGRAMMING TECHNIQUES

When faced with a major system programming task, one must decide to begin coding immediately or construct tools for the job. One such tool is an adequate time-sharing system. This was too big a task to undertake as a part of this thesis, however.

Another tool for systems programming is the construction of a language and a compiler more suited for graphics and data base manipulation than those commonly available. It is not intended to use the design system on the 1108 forever, and the construction of a special programming language, and the generation of its compiler via a meta compiler, are steps which are facilitating the move to another machine.

Graphics Fortran

A "Graphics Fortran" has been constructed as an addition to Fortran V. The Fortran language was chosen strictly on pragmatic grounds and not for its elegance as a computer programming formalism. Specific reasons are:

1) The Fortran compiler is the best maintained compiler on the 1108 and PDP-10 computers.

2) The Fortran compiler on the 1108 produces very efficient code even approaching the quality of hand coding.

3) Utah's Tree-Meta system currently produces Fortran output to take advantage of the Fortran compiler's optimization. The Graphics Fortran Compiler was produced using the Tree-Meta Compiler Generator system.

The Graphics Fortran compiler was not hand-coded in the old, laborious fashion of compiler writers but rather was generated by the Tree-Meta compiler generator system (figure 28). Therefore, the compiler is easy to regenerate whenever a change in the language is desired. The following sketch outlines the process of system generation.

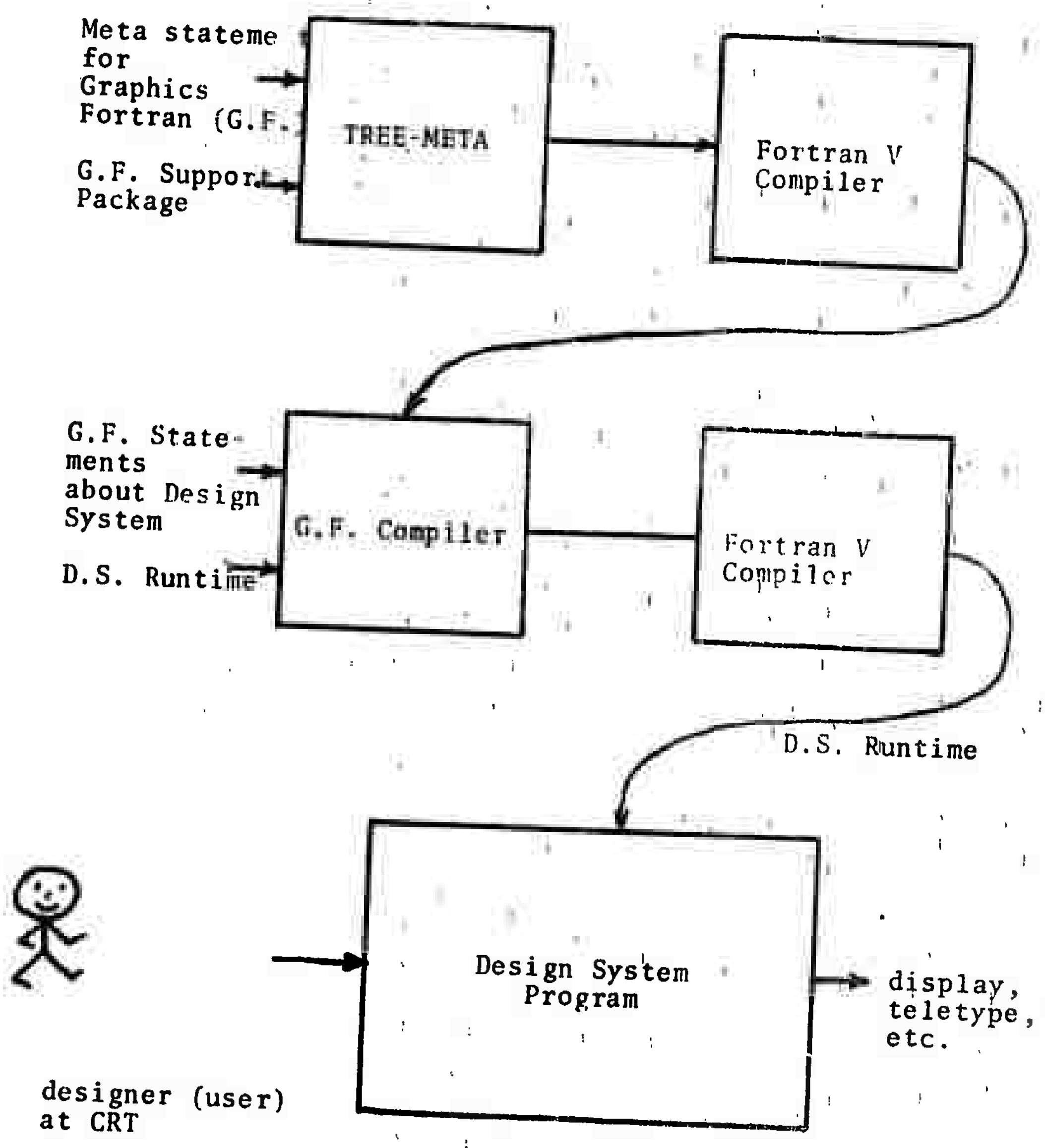
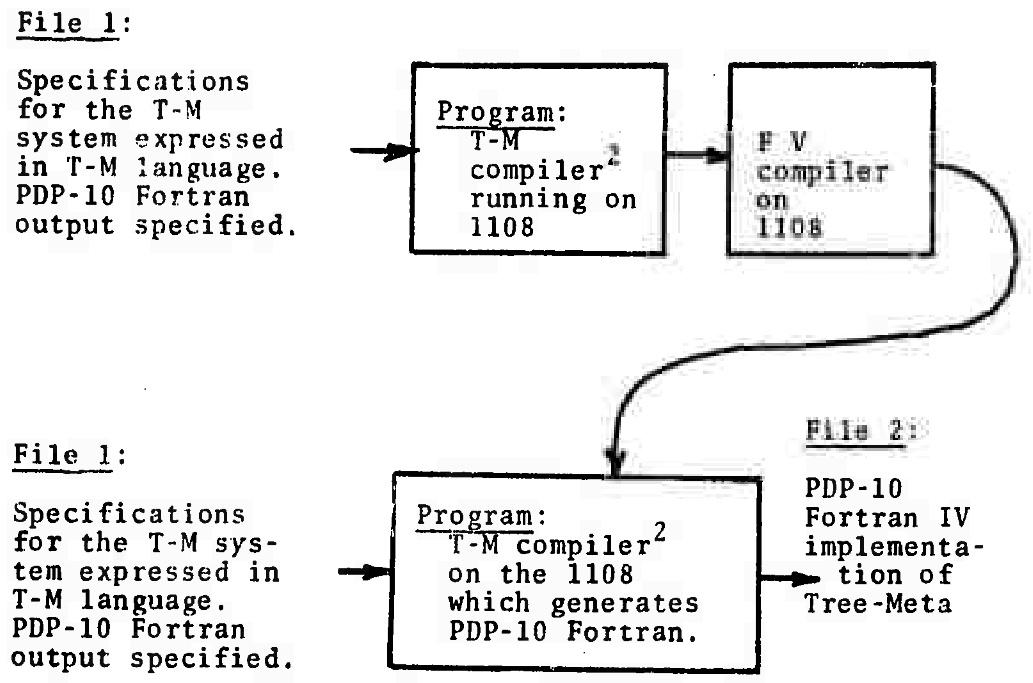


Figure 28. System generation procedure.

Tree-Meta has another important advantage. By a well-known, two-step process, it is possible to move a compiler/compiler to a new machine with a minimum of hand coding (figure 29). Once Tree-Meta is operational on a new machine, compilers such as Graphics Fortran can more easily be moved across to the new machine. <u>The use of Tree-Meta makes moving the Architectural Design System to the PDP-10 vastly easier</u>.


File 1:

Specifications for the T-M system expressed in T-M language. PDP-10 Fortran output specified.

Program: T-M compiler[2] running on 1108

F V compiler on 1108

File 1:

Specifications for the T-M system expressed in T-M language. PDP-10 Fortran output specified.

Program: T-M compiler[2] on the 1108 which generates PDP-10 Fortran.

File 2:

PDP-10 Fortran IV implementation of Tree-Meta

Figure 29. The two-step bootstrapping process to move coding from one computer to another.

Supporting subroutines, such as the identifier recognizer, must be recoded by hand just once when moving between machines.

Graphics Fortran (G.F.) has eight sections or statement types:

1) Regular Fortran statements
2) Associative statements
3) Interaction parsing statements
4) Graphics output statements
5) Segment statements
6) Graph following statements
7) Software interfacing statements
8) Debugging, tracing and statistics gathering, and other miscellaneous statements.

<u>Associative Data Base</u>

Various techniques exist for storing trees within a computer The common methods involve a region of storage divided up into small blocks representing the various nodes. Pointers (addresses or indices) within these blocks indicate connections (branches) to other nodes. Such schemes have serious limitations for, among other things, no finite core size, selected beforehand, is adequate to model all objects. The pointers are of a fixed size and addressing across various levels of storage with them is costly.

As an alternative, the design system presented here stores the relationship of branch/dominant node/dependent node as an associative triple by a method developed at Lincoln Laboratories.[2] Associative triples consist of an attribute/object/value triple. Names are used instead of pointers, as they are independent of the size of the data base.

The names are hashed in various ways such that, given one or two of the triples' components, the third member can be found quickly.

The Lincoln scheme was varied slightly to make it more economical. Graphics Fortran allows the programmer to indicate which components of the triple will be used for future retrieval. In the design system program, retrieval requests are often made in only one or two ways. If the programmer has this information ahead of time, he can save the computer system considerable time and space by indicating it to the machine.

Appendix III

MODIFICATIONS TO THE HIDDEN LINE ALGORITHM

Warnock's hidden line program[10] has been modified to produce output for a line drawing scope, as well as a raster display. The method involves tagging generated points in a way that a high-speed sort results in sets of points which form lines instead of sorting by Y and X coordinate values for raster scope output. Each set of points is replaced by the line (end points) they represent. Generated edges of interpenetrating surfaces were a problem, since they exist only implicitly in the raster output. A technique was devised for detecting such an edge and replacing it with an explicit line segment.

Warnock's implementation was intended for batch processing use, and as such generated all visible edge points before displaying any of them. The modified line drawing algorithm works similarly. One drawback of this method is the intermediate swell of data. One thousand words of surface information becomes 15,000 words of edge point data before it is reduced to 1,000 words of line and point information.

This is not a fundamental problem of the algorithm, of course, for it could be recoded as a set of co-routines to convert generated points to lines as they become available. Coalescing points into lines on the fly would increase the amount of processing done, but would save space (the fundamental time/space trade-off at work, as always).